Making Federated Learning Accessible to Scientists: The AI4EOSC Approach

Judith Sáinz-Pardo Díaz*
sainzpardo@ifca.unican.es
Instituto de Física de Cantabria (IFCA), CSIC-UC
Santander, Spain

Andrés Heredia Canales
heredia@ifca.unican.es
Instituto de Física de Cantabria (IFCA), CSIC-UC
Santander, Spain

Ignacio Heredia Cachá
iheredia@ifca.unican.es
Instituto de Física de Cantabria (IFCA), CSIC-UC
Santander, Spain

Viet Tran
viet.tran@savba.sk
Institute of Informatics, Slovak Academy of Sciences (IISAS)
Bratislava, Slovakia

Giang Nguyen
giang.nguyen@savba.sk
Institute of Informatics, Slovak Academy of Sciences (IISAS)
Bratislava, Slovakia

Khadijeh Alibabaei
khadijeh.alibabaei@kit.edu
Karlsruhe Institute of Technology (KIT), Scientific Computing Center
Eggenstein-Leopoldshafen, Germany

Marta Obregón Ruiz
obregonm@ifca.unican.es
Instituto de Física de Cantabria (IFCA), CSIC-UC
Santander, Spain

Susana Rebolledo Ruiz
rebolledo@ifca.unican.es
Instituto de Física de Cantabria (IFCA), CSIC-UC
Santander, Spain

Álvaro López García
aloga@ifca.unican.es
Instituto de Física de Cantabria (IFCA), CSIC-UC
Santander, Spain

ABSTRACT

Access to computing resources is a critical requirement for researchers in a wide diversity of areas. This has become even more important with the rise of artificial intelligence techniques through the training of machine learning and deep learning models. In this sense, the AI4EOSC project aims to respond to this need by delivering an enhanced set of advanced services and tools for the development of artificial intelligence, machine and deep models, such as federated learning, in the European Open Science Cloud (EOSC). Federated learning is a technology in the field of privacy-preserving machine learning techniques that has revolutionized the current state of the art, evolving from classical centralized approaches to allow training models in a decentralized way, without sharing raw data. In this work, we present the production implementation of a federated learning system based on the Flower framework that allows users, without a technological background, to exploit this technique, performing federated learning training within the AI4EOSC platform. The objective is to be able to train this type of architecture in an intuitive way; for this purpose, a user-friendly dashboard has been implemented, whose development will be reviewed. The frameworks and technologies used for this implementation will be exposed together with an example of use from scratch, in order to demonstrate the use of this functionality of the platform. Finally, two scenarios concerning client availability are analyzed.

*Corresponding author.

IH&MMSec '24, June 24–26, 2024, Baiona, Spain

ACM ISBN 979-8-4007-0637-0/24/06
https://doi.org/10.1145/3658664.3659642

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Artificial intelligence**; **Distributed computing methodologies**; • **Software and its engineering** → **Software development process management**; **Collaboration in software development**.

KEYWORDS

federated learning, privacy-preserving, open science, software development, artificial intelligence

ACM Reference Format:
Judith Sáinz-Pardo Díaz, Andrés Heredia Canales, Ignacio Heredia Cachá, Viet Tran, Giang Nguyen, Khadijeh Alibabaei, Marta Obregón Ruiz, Susana Rebolledo Ruiz, and Álvaro López García. 2024. Making Federated Learning Accessible to Scientists: The AI4EOSC Approach. In *Proceedings of the 2024 ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec '24), June 24–26, 2024, Baiona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3658664.3659642

1 INTRODUCTION

The European Open Science Cloud (EOSC) [15] is an initiative of the European Commission aimed at fostering the creation of a "Web of FAIR Data and Services" for Science in Europe, focused on promoting open science practices among researchers by offering a comprehensive environment for the sharing, processing, and analysis of research data. This initiative overreaches pan-European research infrastructures and services supported by the different EU stakeholders (including the research communities) to support scientific endeavors. A central focus is to strengthen Open Science as a collaborative, transparent, and accessible research model that aims to improve access to scientific resources.

Access to compute and storage resources is often limited, not only in terms of state-of-the-art accelerators (i.e. GPU or TPU), but also in terms of easy access for non-specialized scientists. In

many cases, scientists face the problem of easily exploiting existing resources due to the lack of skills to manage or to deploy complex tasks over the existing e-Infrastructures. This fact becomes worse when the data to be analyzed is not centrally available at a single location and distributed data sources must be exploited. These situations occur when it is complicated or impossible to centralize a large volume of data coming from different sources, sectors, entities, or devices in order to train an artificial intelligence (AI), machine learning (ML) or deep learning (DL) model. In this context, the use of distributed learning (DL) [52] or Privacy-Preserving Machine Learning (PPML) [53] architectures are proving to be valuable solutions. PPML techniques seek to apply machine or deep learning in a trustworthy and secure environment, for example, by enabling decentralized data training to create robust models without the need to share or centralize sensitive data in a single location. Among these architectures, federated learning (FL) [3, 8, 29] stands out for its wide range of acceptance and application.

FL differs from traditional centralized models by redistributing the training process to a decentralized method, addressing concerns about privacy, security, and data sovereignty. The basic principle of this methodology lies in the distributed training of models. Therefore, the architectural framework consists of a central server and several clients, each representing different data owners. This configuration facilitates collaboration and eliminates the need for direct data exchange. Clients train their models locally and, therefore, do not have to disclose their respective datasets. The models or the parameters that describe them are then securely transferred to the central server. Then, the server combines these models using an aggregation operator. After this merging, the updated model is sent back to the clients. Clients are re-trained iteratively for a predetermined number of rounds. This cyclical process, which must be carefully orchestrated, ensures a seamless and secure exchange of information and promotes collaborative learning without compromising the confidentiality of individual datasets.

In this context, the AI4EOSC project [7] provides a compute platform, comprising a set of advanced services specifically tailored to the development of AI, ML and DL models and applications within the EOSC. The primary objective is to enable researchers and institutions within the EOSC to use advanced AI and machine learning techniques for their scientific work. One of the main activity areas of the AI4EOSC project is to deliver tools that will enable scientists to develop privacy-preserving AI models, based on state-of-the-art federated learning frameworks.

In this work, we present the successful integration of the Flower framework [9] into the AI4EOSC platform and provide a comprehensive overview of the challenges encountered during the implementation process. We also discuss the development of various extensions that enable authenticated connections between clients and the server and take into account important aspects of security and data protection. Our exploration of the implementation challenges demonstrates the adaptability of the Flower framework in the AI4EOSC context and highlights the different modifications carried out to achieve the integration.

Finally, to illustrate the practical benefits of our extended implementation, we present a concrete use case in the field of medical imaging. This example shows how Flower, within the AI4EOSC platform, can effectively contribute to federated learning scenarios in real-world applications, especially in sensitive and data-intensive fields such as medical research. In additio, two scenarios in which we deal with intermittent clients are exposed and analyzed.

The reminder of this paper is organized as follows. Section 2 provides an overview of the related work in the areas of federated learning architectures and frameworks, together with different state-of-the-art platforms for computing and training machine and deep learning models. In Section 3 we present the architecture of the AI4EOSC platform with special attention to the components involved, the dashboard and the Application Programming Interface (API) implemented. Section 4 is divided into four key subsections: first, Subsection 4.1 outlines the process of integrating Flower into the platform, Section 4.2 addresses the challenges associated with running the federated learning server behind a proxy, Subsection 4.3 presents the implementation of client authentication and the integration of a secret management system and Section 4.4 provides an illustrative example of a federated learning use case with the server deployed within the AI4EOSC platform, particularly in a medical scenario. Finally, Section 5 draws the conclusions from this work, accompanied by insights into possible future directions of research and development.

2 RELATED WORK

The data science ecosystem and its applications in both industry and research scenarios have grown rapidly in recent years. This is in part due to the availability of computational resources on demand, as well as the ease of use of some IDEs such as JupyterLab or Visual Studio Code (VSCode). JupyterLab is an interactive web-based development environment for notebooks, code, and data [28]. In some contexts, the development of machine learning and data mining processes is increasingly carried out on Jupyter notebooks, mainly because of the ease of use and reproducibility and because of the advantages and convenience in visualization, since it is a dynamic interface. Google Colaboratory, for example, is a service hosted by Jupyter Notebook that provides free access to (limited) computing resources [21]. The disadvantages of coding using Jupyter Notebooks are related to the version control process, but also to the lack of modularity of the code in some cases. Another widely used IDE is Visual Studio Code (VSCode), a Microsoft-developed source code editor [33]. It includes features that are not available in Jupyter, such as debugging support, integrated version control with Git, syntax highlighting, and code refactoring, among others. Note that VSCode includes plugins that allow users to edit and run Jupyter notebooks.

Open Science Environments (OSEs) seek to promote collaborative and transparent scientific research by providing access to computational resources, diverse tools, and cloud services. For example, Terra [18] serves as a cloud-based platform specifically designed for biomedical research and genomics. It provides researchers in these fields with a seamless environment for analyzing, sharing, and collaborating on data. Dataone [10] is dedicated to enhancing access to and sharing of environmental and geoscience data. By facilitating collaboration and data sharing, Dataone contributes to a more accessible and interconnected research ecosystem in these fields. CyVerse [11] offers a suite of computational tools tailored to life science research. The comprehensive set of resources supports

data-driven investigations and provides researchers with a platform for conducting analysis, collaboration, and innovation in the life sciences. Open Science Grid (OSG) [37] specializes in creating a distributed computing infrastructure tailored for high-throughput computing (HTC). Researchers use the OSG to efficiently perform large-scale computations that enable complex simulations and data-intensive analysis. The DEEP Hybrid DataCloud framework [32] contributes to the entire data lifecycle with computational resources and cloud services. Researchers benefit from a set of tools to manage, analyze and extract insights from their data across diverse domains.

In line with the OSEs concept, AI4EOSC provides users with both computational resources and a range of tools to optimize the implementation and further development of machine learning and deep learning techniques. In particular, AI4EOSC supports advanced functionalities for distributed and federated learning. Within this framework, distributed learning involves machine or deep learning on decentralized data, addressing considerations such as privacy, legal requirements, technical constraints, and more.

The increasing need to comply with various privacy schemes and the existence of technical restrictions in certain use cases have driven the development and application of machine and deep learning models on distributed data. This demand has led to the widespread adoption of federated learning architectures in various domains. In a classic federated learning architecture, a server orchestrates the collaboration of multiple clients to train a shared model without the need to exchange raw data. The server collects the locally trained models of the individual clients and aggregates them into a global model. This iterative process takes place over a predefined number of rounds, during which the server sends the updated model to the clients for local re-training, employing a predefined aggregation strategy.

The applications of federated learning extend to various domains where privacy and technical constraints are critical considerations. In the field of medical imaging, federated learning has shown promise for future advances [34, 45, 48]. The industry has also embraced federated learning for collaborative model training while maintaining data privacy [35, 36]. Furthermore, federated learning is relevant in environmental sciences, where it addresses the challenges associated with distributed data [14, 17, 47].

Various software frameworks have emerged in the FL space that facilitate the development and deployment of FL models in various domains. Notable examples include TensorFlow Federated (TFF), based on TensorFlow [50]; PySyft [38], which uses PyTorch; FATE (Federated AI Technology Enabler) [16] hosted by Linux Foundation; NVIDIA Flare [46]; and Flower [9]. Regarding the first, TensorFlow Federated [50] is an architecture-agnostic open-source framework for ML and other computations on decentralized data from the TensorFlow family. PySyft [38], from OpenMined's open-source stack, allows users to apply DL models with privacy-preserving functionalities, being in active development. FATE [16] is an open-source framework that provides support for federated learning architectures and secure computation in ML ecosystems. NVIDIA FLARE [46] from NVIDIA, allows moving from a centralized ML or DL approach to a federated one, it is open-source and domain agnostic.

Following a comprehensive evaluation of various framework options during the project, Flower emerged as the most compelling choice for integration into the AI4EOSC platform due to several key features:

(1) Ease of use: Flower provides a high-level API that facilitates the implementation of federated learning systems.
(2) Flexibility with various DL frameworks such as PyTorch and TensorFlow.
(3) Active community and development.

The AI4EOSC platform provides users with a federated learning server that enables federated training with clients deployed on external cloud machines, locally, or within the same platform in different deployments. This federated server, which is integrated as a tool in the AI4EOSC dashboard, simplifies the server implementation process. Users can initiate the server effortlessly, facilitating seamless connection with different clients and enabling the start of the training processes. To our knowledge, there are currently no OSEs that integrate the Federated Learning (FL) Server as a dedicated tool in their framework.

3 AI4EOSC PLATFORM

The AI4EOSC platform is built as a software ecosystem designed to enhance the development and deployment of artificial intelligence, machine learning, and deep learning applications within the context of the European Open Science Cloud (EOSC) [7]. As a Platform-as-a-Service (PaaS) cloud computing model, both hardware and software resources are managed to provide an environment in which users can focus solely on application development. To achieve reliable, efficient, and secure performance, the platform leverages open-source software solutions. These comprise the use of Consul [24], Nomad [25] and Traefik [30], all of which are seamlessly integrated.

Consul offers secure connectivity, service discovery, and health monitoring to deploy a centralized and automated service network. Through Consul, several on-premise cloud instances with diverse configurations (i.e. CPUs, GPUs, volumes sizes) are automatically registered and connected to support the computing requirements of the different AI4EOSC users. Once the cloud network is set up, Nomad integrates with it to manage the scheduling and orchestration of the application workloads across the available nodes to ensure efficient resource allocation. By connecting Nomad to Consul, every service deployed in Nomad as a job is automatically registered on Consul through its service discovery feature. This allows the Traefik instance, which is easily deployed as a dedicated Nomad job, to provide routing for all the healthy services registered in the Consul catalog. Specifically, Traefik acts as a reverse proxy that exposes jobs under the specific AI4EOSC domain.

When a user accesses the AI4EOSC dashboard[1] to launch an application workload, it is transparently submitted as a Nomad job through a platform API [6] request. Successfully scheduling a job means filtering all available nodes to match the computational requirements of the job or deployment. Nomad is responsible for selecting a node from the eligible subset of the cluster based on load-balancing policies to allocate the job. Once the job is allocated, it automatically starts to run. Then, the user can access their deployed

[1]The AI4EOSC dashboard is accessible through https://dashboard.cloud.ai4eosc.eu.

jobs on the URL provided by Traefik. A universal unique identifier (UUID) is associated with each deployment.

3.1 AI4EOSC API

As can be understood from the previous description, the AI4EOSC platform API [6] provides a single entry point to access all components of the platform. It hides the intrinsic complexity of the underlying technologies used and provides users with a simple interface to use the services, making it possible to replace the underlying orchestration technology without modifying the platform API semantics.

The API is written in Python [51], using FastAPI [44] for the API logic and the Nomad Python client [27] to deploy jobs in the cluster. It follows the REST model and automatically generates an OpenAPI specification as long as a Swagger interface [1] interacts with its endpoints. It also follows DevOps best practices, passing integration tests before pushing.

The API is currently organized around two main routers: `/catalog` and `/deployments`. The catalog router handles all the functionalities to get information about AI modules and tools. The deployments router handles the functionality to manage deployments of AI modules (and tools) in the Nomad cluster.

In the context of the federated learning server, the API allows to retrieve the default configuration parameters (described in Section 4.1). Then, it takes the user-provided custom settings to configure a special Nomad job tailored to the FL server use case that will be sent to the cluster.

The Nomad job configures the ports available to the FL server, the domain to access the services via Traefik, it enables the use of gRPC in the connections and finally sets the appropriate hardware resources needed.

For each federated server, the API automatically generates a secure hexadecimal token. This token is then saved in the project's Vault instance [2] and will be used to authenticate the federated clients wishing to join the common training (additional tokens can be created if needed). All federated server endpoints are protected with SSL.

Only properly authenticated users (via EGI Check-In) are allowed to deploy a FL server. Once created, the API allows the user to list all his/her currently deployed servers and to delete them.

3.2 AI4EOSC Dashboard

The AI4EOSC Dashboard [41] [42] is a single web application, developed using modern web development frameworks: namely Angular [19] and Jest [26] for the business logic and the Material Angular UI component library [20] to create a uniform and user-friendly interface.

Following software engineering and Angular good practices, the dashboard is build using individual components that are tested and can be reused to reduce code duplication and thus improving the maintainability of the project. On top of this, modern DevOps techniques and platforms are being used to enable a reliable, reproducible, and fast way to integrate new features into it. Platforms such as Github with the use of Github actions are relied upon to perform several jobs, such as building docker images and uploading them to a private Harbor registry [23] or to use the release-please library [22] to create automated releases.

The dashboard enables users to interact with the platform in a simple and graphic way. Once logged-in, its divided into two main sections: (1) Marketplace and (2) Deployments.

The first section allows users to launch deployments based on either a generic base module that is used as a baseline to create new modules, or one of the wide list of ready-to-train modules. A tool, in contrast to a ready-to-train module that often comes with a pre-trained model, is meant to provide an environment with a software toolset aimed at solving a problem that fits it; therefore, they are treated differently in the dashboard as they need special configuration inputs that are tailored for each of them. This first section is available for registered and unregistered users, although with limited options. All the modules and tools listed in this sections have a detailed view that shows additional information such as a brief description, the categories it belongs to, links to the code repository, the docker image, and even the dataset (if available). In Figure 2 an example of the AI4EOSC marketplace with the different pre-trained modules is shown.

The second section of the dashboard displays the current deployments of the logged-in user, differentiating between modules and tools so that they can monitor their status in the cluster and perform other actions such as accessing a deployment's main endpoint or deleting the deployment altogether.

To further facilitate user access to the platform, the dashboard is integrated with the EGI check-in Authentication and Authorization Infrastructure (AAI) [13] (see the C4 architecture with respect to the landscape of the system and the EOSC AAI in Figure 1). This means that users can reuse their current credentials used for other EOSC projects and only have to request approval to be part of the AI4EOSC virtual organization before they can start using the dashboard.

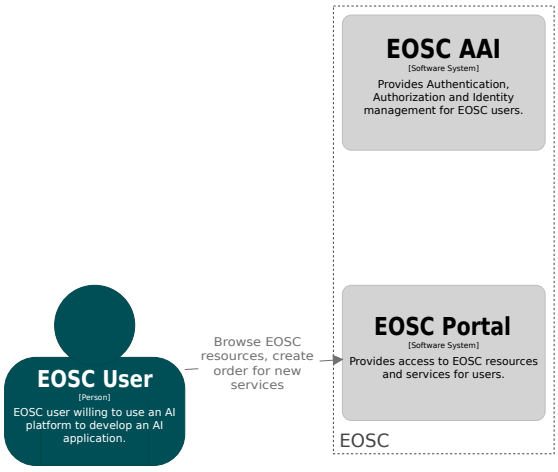

Figure 1: AI4EOSC system landscape and relation with the EOSC, C4 model. Extracted from [40].

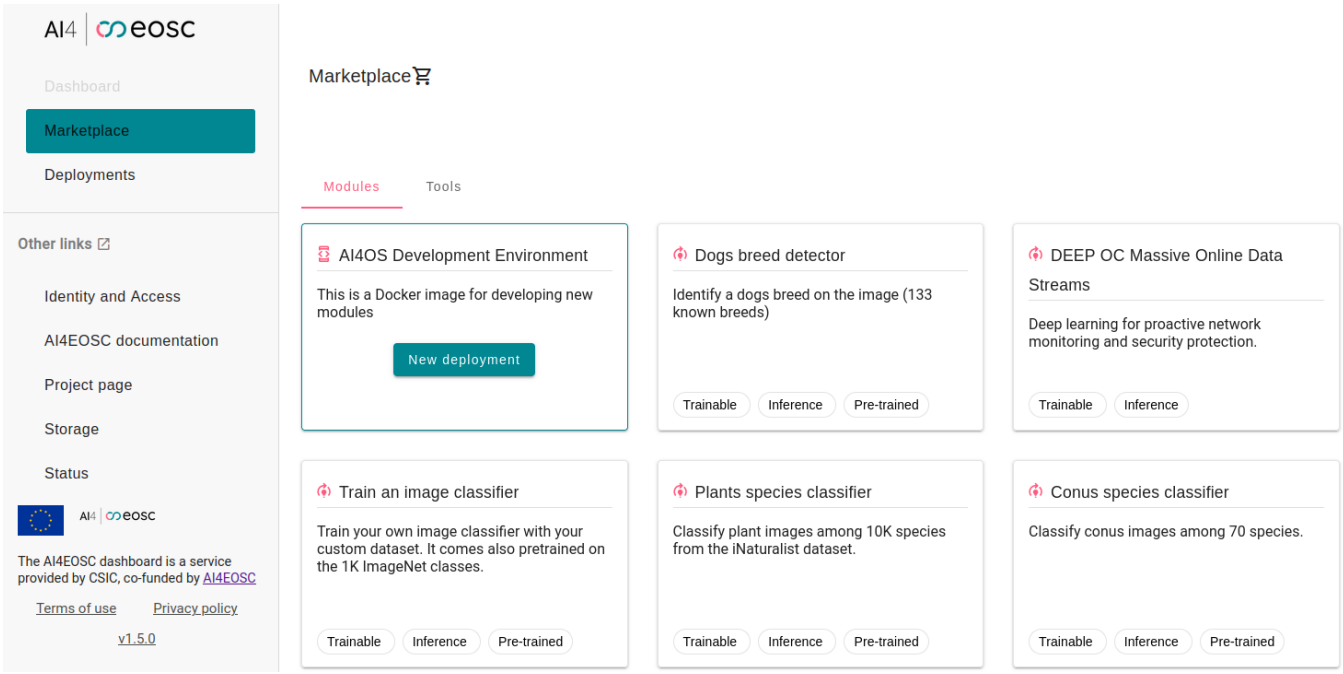

Figure 2: AI4EOSC dashboard: modules.

4 FEDERATED LEARNING IN AI4EOSC

In this section, we present the implementation of a federated server that allows users to perform federated training of machine and deep learning models using the Flower library and the AI4EOSC platform. We describe different issues addressed in relation to the connection of the clients when they are connected through a proxy or load balancer. We also present the implementation of different extensions to allow an authenticated connection of the clients with the server, based on the use of bearer tokens with an efficient secret manager that supports the storage, management and revocation of these tokens. Finally, we present an extensive example of use in a medical imaging scenario.

4.1 Implementation using Flower

As already outlined in Section 2, there are different state-of-the-art frameworks that provide federated learning functionalities for ML and DL frameworks. Some of the most widely used in the field are *TensorFlow Federated*, *PySyft*, *FATE*, *NVIDIA FLARE* or *Flower*, among several others. As explained in Section 2, the final choice of the framework used was mainly based on a search for a balance between functionality, efficiency, and user-friendliness. In this sense, the Flower library [9], which claims to be *a Unified Approach to Federated Learning* was chosen. The Flower library is framework-agnostic and its implementation allows for easy extensibility, being under active development by an open community. It allows moving from a centralized to a federated approach in a simple and intuitive way and just in a few lines of code, without extensive modifications on the original, centralized, code base.

The AI4EOSC platform has therefore adopted Flower to provide users with a federated learning server to allow distributed training with clients deployed in external cloud machines, locally in their own workstations, or on the AI4EOSC platform itself. The so-called federated server has been implemented and incorporated as a tool within the AI4EOSC dashboard, and provides the implementation of the server so that users only have to run it to start connecting the different clients and performing the training. To do this, when the tool is started, different configuration parameters must be set from the dashboard (or API) in a very straightforward way. There are three steps to configure the federated server:

- *General configuration:* configuration regarding the deployment options, the service to run, and the docker options. The different fields to be completed are:
  - *Deployment options:* deployment title (less than 45 characters), deployment description, custom domain for the endpoint (if needed) and service to run (fedserver, JupyterLab or Visual Studio Code, that can be spawned for debugging purposes on the same node). Moreover, the federated server is configured with a Bearer token for authentication, allowing the user to setup additional tokens if needed (for example, to share with different users). In order to do so, the server takes advantage of the authentication modules provided by the `ai4-flwr` library [5], developed within AI4EOSC and which will be explained in 4.3. Each secret will be associated with a unique identifier in case it needs to be revoked. These secrets will be stored using Vault as will be explained in Section 4.3.
  - *Docker options:* docker image and docker tag to use for deployment, only for advanced users.
- *Hardware configuration:* specifically in this type of deployments no special hardware requirements are necessary, as the model training is done on the clients' side, and they will have the highest

computational requirements, for example in terms of RAM, CPU and GPUs.
- *Federated configuration:* is the most important part, as it includes the specifications that the federated training must satisfy. Specifically, the parameters to be filled in by the user are the following.
  - *Federated rounds:* number of rounds that the training process will be repeated.
  - *Minimum number of clients:* number of clients that must be connected to the server and ready to train the models to start the process.
  - *Federated metric:* is the metric used for monitoring and validating. More than one metric can be included.
  - *Federated aggregation strategy:* is the function or strategy that the server will apply to aggregate the models received from the clients. It must be selected among different predefined options: Federated Average, FedProx strategy, Adaptive Federated Optimization using Adam, Adaptive Federated Optimization, and federated optim strategy Yogi.

This opinionated set of parameters to be setup by the user hide the underlying configuration of the server, providing users with just the functionality that is needed to start a federated server.

Once the server is started (either automatically or via Visual Studio Code or JupyterLab) it is waiting for the clients to connect in order to start the training, as it can be seen in Code 1, also with the token interceptor deployed using the secrets stored in Vault (in this case only with one token).

```
vault.py:76 | Configured Vault Bearer token
              authentication with:
              'https://vault.services.fedcloud.eu:8200/'
vault.py:77 | Reading tokens stored in:
              'users/.../deployments/.../federated'
Getting tokens from Vault ->
            users/.../deployments/.../federated
vault.py:79 | Configured Vault Bearer tokens:
              ['a7fa98...']
vault.py:120 | Renewing Vault token...
vault.py:126 | Vault token is valid for 43199 seconds
server.py:80 | Token interceptor created
app.py:167 | Starting Flower server, config:
             ServerConfig(num_rounds=10,
             round_timeout=None)
app.py:181 | Flower ECE: gRPC server running
             (10 rounds), SSL is disabled
server.py:89 | Initializing global parameters
server.py:276 | Requesting initial parameters
                from one random client
```

Code 1: Starting the federated server (monitoring).

The federated server implementation is located in [43]. Specifically, the server code starts a Flower server with the specifications given by the user when creating the deployment. It should be noted that [39] present some modifications to the original implementation of the Flower library in order to be able to run behind a proxy and to authenticate the clients using a token, as will be explained in the following sections.

The AI4EOSC architecture is based on the C4 model [40]. In Figure 3 the architecture of the federated learning scenario is shown.

4.2 Running the server behind a proxy

As already mentioned, in order to effectively manage the distribution of resources according to the requests of the platform users, on the AI4EOSC platform we use *Traefik* as a proxy or load balancer.

It is important to note that the connection of the different clients to the FL server using Flower is performed by means of gRPC (gRPC Remote Procedure Call), which facilitates efficient communication between distributed applications. Specifically, a gRPC server is deployed to carry out this connection. The gRPC server is responsible for listening to the client requests and responding to them according to the specified service definitions. If the gRPC server is running behind a load balancer (as in our case, Traefik), clients may not be able to connect to the server. In version 1.6.0, Flower is using the *peer()* method from *grpc.ServicerContext* in order to identify unique Flower clients and avoid duplicate clients. However, in some situations (like when running the gRPC server behind a load balancer or a proxy), different clients can have the same peer identifier (in this case corresponding to the same IP port), as HTTP/2 connections are multiplexed (gRPC uses HTTP/2 as transport protocol).

The solution that we proposed for this issue has already been included in version 1.7.0 of the Flower library.

4.3 Client authentication

When working with FL architectures where multiple parties are involved (one server and multiple clients), it is important to maintain the integrity and security of the process. In this case, we assume that the server is honest, i.e., it is supposed to be trustworthy and to faithfully follow the protocol. In other cases, we may deal with honest-but-curious server (which might extract privacy from the model updates sent by each client [31]) or even malicious servers.

In the original implementation of the Flower library, clients only need to enter the endpoint of the deployment where the server is being deployed to connect to it. In a typical case, it would be enough to type in the IP address and the port where the server is deployed. However, this can lead to certain security problems if external attackers attempt to enter federated training for malicious purposes. For example, what are known as poisoning attacks consist of affecting the learning model during the training process. This could be done by a client that enters deliberately wrong labels for its data in order to damage the overall accuracy of the model, by introducing other kind of data for training, or just by sending random weights or parameters for the model.

For this reason, we have implemented an authentication system for clients prior to their incorporation into the federated training through the use of tokens.

The tokens themselves are considered as secrets and must be managed in a secure and systematic manner. In the event of a security breach where the secret or token is exposed, it would be revoked, and there would be no direct way to modify the server to accept another secret. Therefore, the implementation of a secret management service, specifically HashiCorp's Vault [2], has been adopted for storing the tokens. The federated learning server retrieves a list of authorized tokens for client authentication from

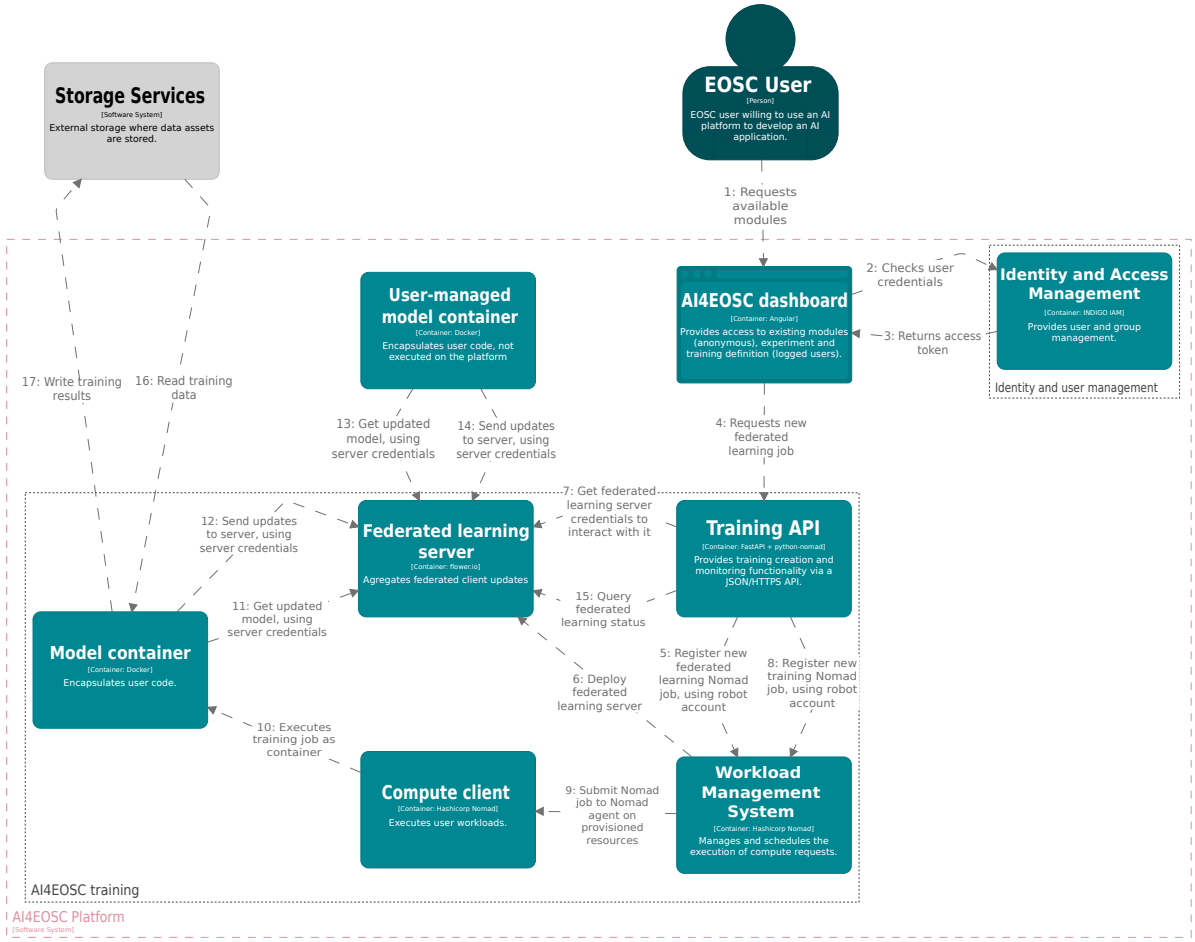

Figure 3: AI4EOSC federated training architecture, C4 model. Extracted from [40].

the secret management service and verifies if the client's provided token is included. Users can manage tokens (add, rotate, or revoke) through the service's native command-line client or graphical user interface. A sidecar process monitors the secret management service and notifies the federated learning server to update its token list whenever changes occur.

Currently, we have implemented some extensions to the Flower library to work alongside the Vault service that stores the secrets (it can be found in [5]). In addition, we have developed some modifications to the Flower library to manage the credentials that allow to enable or disable the connection to the clients based on the token entered (see [4]).

In Code 2 we show how the token interceptor is created using the secrets stored in Vault and how the server is started including the interceptor. Note that the strategy and the other federated settings (e.g. number of rounds) are generated using the information provided when creating the deployment.

```
UUID: str = os.environ['NOMAD_JOB_NAME'][8:]
USER: str= os.environ['NOMAD_META_owner']
VAULT_TOKEN: str = os.environ['VAULT_TOKEN']
(...)
from ai4flwr.auth.vault import VaultBearerTokenInterceptor
# Include token interceptor
path_deployment = f"users/{USER}/deployments/{UUID}"
token_interceptor = VaultBearerTokenInterceptor(
    vault_addr="https://vault.services.fedcloud.eu:8200/",
    vault_token=VAULT_TOKEN,
    vault_mountpoint="/secrets/",
    secret_path=f"{path_deployment}/federated"
)
# Start the server
fl.server.start_server(
    server_address="0.0.0.0:5000",
    config=fl.server.ServerConfig(
        num_rounds=FEDERATED_ROUNDS
    ),
    strategy=strategy,
    interceptors=[token_interceptor],
)
```

Code 2: Flower server with token interceptor.

It is important to highlight that authentication will be useful to prevent unauthorized clients from participating in training for malicious purposes. However, it will not prevent poisoning attacks in case an authenticated and trusted client misbehaves and sends misleading weights. However, if this happens and the server has a module capable of detecting anomalous behavior in a client, it will be able to revoke the token that such client used to authenticate.

4.4 Example from scratch: medical imaging

We will now proceed to exemplify how to carry out a FL training using the AI4EOSC platform. Specifically, we will use the data and the model presented in [49], where a case study of medical imaging is proposed. More specifically, we are dealing with a classification problem where, given an chest X-ray image, the goal is to classify based on whether the patient has pneumonia or not. The data used come from a classical open dataset, available in [12]. To apply the federated learning architecture, the idea is to distribute the data available in the training set to three different clients, so that we simulate that we have three different hospitals seeking to collaborate in order to build a global and robust DL model for this purpose. The model applied to the three clients is an ANN composed of the layers exposed in [49].

The first client analyzed contains 1050 images for training and 350 for testing, the second one 1800 for training and 600 for testing, and the last one 1062 for train and 354 for test. Figure 4 shows one random image selected from each client train set from a patient with pneumonia and another corresponding to a patient which do not present pneumonia.

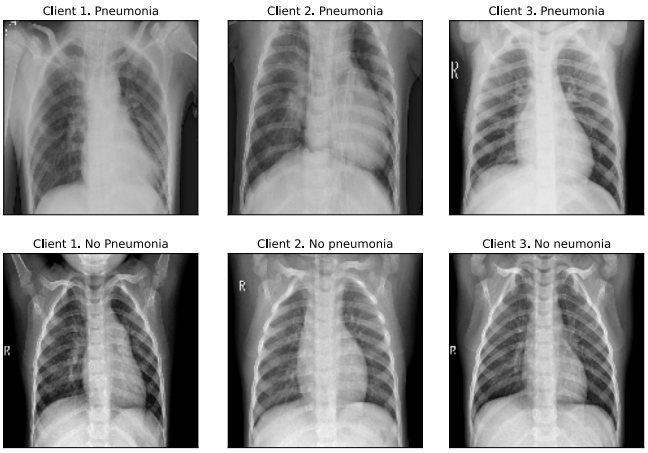

Figure 4: Example of training images for each client.

The following federated learning configuration will be applied: *number of rounds: 10, federated metric: accuracy, minimum number of clients: 3, federated strategy: federated average.*

In order to perform the federated training, we have deployed three instances at IFCA's cloud using openstack in order to allocate and run the data and the code of the three different clients. All these three machines are provided with Ubuntu 22.04, 4 CPUs cores, 10.74GB of RAM and 20GB of disk memory. The server was deployed within the AI4EOSC platform with the federated specifications given above.

In each of them we have used a virtual environment with Python 3.10 and we have deployed the FL clients. In the code given in Code 3 we can see the example for the first client, in which we first define the class *Client1*, which inherits from the NumPyClient class of flower.client, and then we start the connection from the client to the server using the endpoint of the last one (see Code 4). Note that in this example the uuid of the deployment where the server is hosted is *5d46d419-d6de-11ee-bbe3-a0cec839f955*.

```
class Client1(fl.client.NumPyClient):
    def get_parameters(self, config):
        return model.get_weights()

    def fit(self, parameters, config):
        model.set_weights(parameters)
        model.fit(x_train, y_train, epochs=5,
                  batch_size=16)
        return model.get_weights(), len(x_train), {}

    def evaluate(self, parameters, config):
        model.set_weights(parameters)
        loss, accuracy = model.evaluate(x_test, y_test)
        return loss, len(x_test), {"accuracy": accuracy}
```

Code 3: Class Client1.

```
# Import BearerTokenAuthPlugin from ai4flwr:
from ai4flowr.auth.bearer import BearerTokenAuthPlugin
token = "a7fa98..."
auth_plugin = BearerTokenAuthPlugin(token)
# UUID of the deployment of the FL server (AI4EOSC):
uuid = "5d46d419-d6de-11ee-bbe3-a0cec839f955"
fl_uuid = f"fedserver-{uuid}"
end_point = f"{fl_uuid}.deployments.cloud.ai4eosc.eu"
fl.client.start_client(
    server_address=f"{end_point}:443",
    root_certificates=Path(certifi.where()).read_bytes(),
    client=Client1().to_client(),
    call_credentials=auth_plugin.call_credentials()
)
```

Code 4: Connecting the client with the server deployed in AI4EOSC.

First, we initialize the server as described in Code 1. Then, when we run the first client connecting it to the server, the monitoring status on the server side is updated as follows (see Code 5):

```
app.py:167 | Starting Flower server, config:
             ServerConfig(num_rounds=10,
             round_timeout=None)
app.py:181 | Flower ECE: gRPC server running
             (10 rounds), SSL is disabled
server.py:89 | Initializing global parameters
server.py:276 | Requesting initial parameters
                from one random client
```

```
server.py:280 | Received initial parameters
                from one random client
server.py:91 | Evaluating initial parameters
server.py:104 | FL starting
```

Code 5: Starting the FL server and connecting the first client.

The status has now changed to FL starting, however, the process will not begin until the minimum required number of clients have been connected. From the client side, we can see how the gRPC connection has been established and is ready to start training once the rest of the clients (the minimum number) have connected and therefore the server allows it:

```
bearer.py:125 | Created AuthMetadataPlugin
                with token: a7fa98...
grpc.py:65 | Opened secure gRPC connection using
             certificates
connection.py:58 | ChannelConnectivity.IDLE
connection.py:58 | ChannelConnectivity.CONNECTING
connection.py:58 | ChannelConnectivity.READY
```

Code 6: Connecting the first client with the server.

Once the minimum number of clients have been connected (in this case three), the training of the model locally on each client will start, and will be repeated for the pre-fixed number of rounds. Thus, the evolution of the process from the server side is shown bellow (skipping the intermediate rounds just for better visualization) with the corresponding results as a function of the aggregated metric (in this case the accuracy):

```
server.py:104 | FL starting
server.py:222 | fit_round 1: strategy sampled
                3 clients (out of 3)
server.py:236 | fit_round 1 received 3 results
                and 0 failures
fedavg.py:250 | No fit_metrics_aggregation_fn
                provided
server.py:173 | evaluate_round 1: strategy
                sampled 3 clients (out of 3)
server.py:187 | evaluate_round 1 received 3 results
                and 0 failures
(...)
server.py:222 | fit_round 10: strategy sampled
                3 clients (out of 3)
server.py:236 | fit_round 10 received 3 results
                and 0 failures
server.py:173 | evaluate_round 10: strategy sampled
                3 clients (out of 3)
server.py:187 | evaluate_round 10 received 3 results
                and 0 failures
Only one metric has been entered.
server.py:153 | FL finished in 777.3200013944879
app.py:231 | app_fit: losses_distributed
    [(1, 5.591757377963856), (2, 2.0657108459004596),
    (3, 0.1423788307908854), (4, 0.06921767829660258),
    (5, 0.5391561854287891), (6, 0.17663749540510354),
    (7, 0.06715587471137566), (8, 0.12410825495522447),
    (9, 0.1657278991070079), (10, 0.19349358802192781)]
app.py:232 | app_fit: metrics_distributed_fit {}
app_fit: metrics_distributed
    {'accuracy': [(1, 0.7430981524700037),
    (2, 0.7684049252534937), (3, 0.9616564408036097),
    (4, 0.9754601416229471), (5, 0.9164110243868975),
    (6, 0.9516871268032518), (7, 0.9777607484463534),
    (8, 0.9693251606876865), (9, 0.9662576779448914),
    (10, 0.9616564420834641)]}
```

Code 7: Federated learning training. Monitoring from the server side.

In Code 7 we can see the evolution (from the server side), of the loss and the accuracy distributed and aggregated according to the number of data of each client. While in the first round the distributed loss (aggregated) was 5.5918, this value decreases to 0.1935 in the last round, showing the model convergence. The same applies for the aggregated distributed accuracy, with was 0.7431 after the first round, and 0.9617 after the last one. This process can also be monitored from the clients' side during the training.

Then, for each client test set, we get the results in terms of loss, accuracy, and AUC-ROC (in the following AUC) presented in Table 1 when predicting using the global updated model. Here we can note that the client which has the great amount of data (client 2) presents the best results in terms of both loss, accuracy and AUC, which goes hand in hand with the fact that the weights have been aggregated in a weighted way according to the number of training data for each client.

Table 1: Loss, accuracy and AUC obtained for each test set with the global model.

CLIENT	*Loss*	*Accuracy*	*AUC*
Client 1	0.1657	0.9629	0.9958
Client 2	0.0577	0.9883	0.9991
Client 3	0.4512	0.9153	0.9888

Thus, the federated training of a deep learning model has been carried out in three clients deployed in three different external cloud instances, with the federated server deployed in AI4EOSC, managing to build a global model without having to share the clients' data at any time. All server configuration has been carried out through the dashboard, being necessary only to execute the code available in this deployment to initialize the server and start the client connection process.

4.5 Example with intermittent clients

Some of the commonly encountered problems when implementing and deploying a federated learning architecture in production are related to the availability of the involved clients. For example, in the study carried out in [49] used as a reference for this analysis, different scenarios are analyzed related to the entry of a new client in the training once it is started or in case a client leaves the training.

The following describes how these casuistics are handled using Flower in AI4EOSC.

One new client joins the training. To analyze the case where a new client enters the training once it has already started, we have created a new FL server on the platform with the same configuration as in the previous case, except for the minimum number of clients, which in this case will be 2. Therefore, we start by connecting clients 1 and 2, and it will not be until training round number 9 that we will connect the third client (specifically during round 8). Code 8 shows the evolution from the server's point of view.

```
server.py:104 | FL starting
server.py:222 | fit_round 1: strategy sampled
                2 clients (out of 2)
server.py:236 | fit_round 1 received 2 results
                and 0 failures
fedavg.py:250 | No fit_metrics_aggregation_fn
                provided
server.py:173 | evaluate_round 1: strategy
                sampled 2 clients (out of 2)
server.py:187 | evaluate_round 1 received 2 results
                and 0 failures
(...)
server.py:222 | fit_round 8: strategy sampled
                2 clients (out of 2)
server.py:236 | fit_round 8 received 2 results
                and 0 failures
server.py:173 | evaluate_round 8: strategy sampled
                3 clients (out of 3)
server.py:187 | evaluate_round 8 received 3 results
                and 0 failures
Only one metric has been entered.
server.py:222 | fit_round 9: strategy sampled
                3 clients (out of 3)
server.py:236 | fit_round 9 received 3 results
                and 0 failures
server.py:173 | evaluate_round 9: strategy sampled
                3 clients (out of 3)
server.py:187 | evaluate_round 9 received 3 results
                and 0 failures
Only one metric has been entered.
server.py:222 | fit_round 10: strategy sampled
                3 clients (out of 3)
server.py:236 | fit_round 10 received 3 results
                and 0 failures
server.py:173 | evaluate_round 10: strategy sampled
                3 clients (out of 3)
server.py:187 | evaluate_round 10 received 3 results
                and 0 failures
Only one metric has been entered.
server.py:153 | FL finished in 642.8026974536479
app.py:231 | app_fit: losses_distributed [
    (1, 6.624357072930587), (2, 0.20727387227510152),
    (3, 0.059018391527627646), (4, 0.2510757273749301),
    (5, 0.25708073220754923), (6, 0.2698057278206474),
    (7, 0.08329900314933375), (8, 0.1921575677758254),
    (9, 0.938873184147788), (10, 6.897786095829829)]
app.py:232 | app_fit: metrics_distributed_fit {}
app.py:233 | app_fit: metrics_distributed
    {'accuracy': [(1, 0.7894736779363531),
    (2, 0.9515789402158636), (3, 0.9800000033880535),
    (4, 0.9463157747921191), (5, 0.9052631698156658),
    (6, 0.9631579173238654), (7, 0.975789487361908),
    (8, 0.971625775281637), (9, 0.9302147407473231),
    (10, 0.7944785255047441)]}
```

Code 8: Federated learning training: intermittent clients (new client enters the training). Monitoring from the server side.

In view of the results observed in Code 8, both in relation to the distributed metric (accuracy) and to the distributed loss, it is immediate to observe the anomaly derived from round 8 to 10. This is due to the fact that in this round client three is connected, which had not participated before in the training, but is included during the evaluation. However, in round 9 and 10 we see a high decrease in the global distributed accuracy and an increase in the loss.

One client leaves the training. Here the idea is to simulate the availability of a client that experiences connectivity issues and therefore connects or disconnects at different times during the training. This is a very common problem in different sectors and can be due to different reasons, from latency, to internet connection problems or computational difficulties that paralyze the training.

It should be noted that if we set 3 as the minimum number of clients both available in the system and for training during the federated configuration, and one of them disconnects, the process will stop until there is a third client connected. Therefore, in this case we initialize a server with 2 as the minimum number of clients, and the rest of the parameters of the FL configuration as in the previous case. Then, we connect for the first round the first two clients, and during the course of this we connect the third one. In this case, client number three leaves the training during round number 9 (once it is started). The process followed from the server side can be seen in Code 9.

```
server.py:104 | FL starting
server.py:222 | fit_round 1: strategy sampled
                2 clients (out of 2)
server.py:236 | fit_round 1 received 2 results
                and 0 failures
fedavg.py:250 | No fit_metrics_aggregation_fn
                provided
server.py:173 | evaluate_round 1: strategy sampled
                3 clients (out of 3)
server.py:187 | evaluate_round 1 received 3 results
                and 0 failures
(...)
server.py:222 | fit_round 9: strategy sampled
                3 clients (out of 3)
server.py:236 | fit_round 9 received 3 results
                and 0 failures
server.py:173 | evaluate_round 9: strategy sampled
                2 clients (out of 2)
server.py:187 | evaluate_round 9 received 2 results
                and 0 failures
```

```
Only one metric has been entered.
server.py:222 | fit_round 10: strategy sampled
                2 clients (out of 2)
server.py:236 | fit_round 10 received 2 results
                and 0 failures
server.py:173 | evaluate_round 10: strategy sampled
                2 clients (out of 2)
server.py:187 | evaluate_round 10 received 2 results
                and 0 failures
Only one metric has been entered.
server.py:153 | FL finished in 667.3757740734145
app.py:231 | app_fit: losses_distributed
    [(1, 7.547033552743175), (2, 0.2451032611764282),
    (3, 0.8357470439621276), (4, 0.1293588922204781),
    (5, 2.1022320605494493), (6, 0.6716479192847854),
    (7, 0.15273705459674078), (8, 0.0717641580392795),
    (9, 0.057207336943400536), (10, 0.5714634751018725)]
app.py:232 | app_fit: metrics_distributed_fit {}
app.py:233 | app_fit: metrics_distributed
    {'accuracy': [(1, 0.7430981524700035),
    (2, 0.9271472321331866), (3, 0.8558282225958409),
    (4, 0.955521475135183), (5, 0.7783742496755225),
    (6, 0.8949386509466757), (7, 0.9647239230893141),
    (8, 0.9746932575673414), (9, 0.9852631688117981),
    (10, 0.9084210615409049)]}
```

Code 9: Federated learning training: intermittent clients (new client enters the training). Monitoring from the server side.

As in the previous case, the inclusion of a client in a round that has already begun is reflected in the loss and accuracy. In this case it is especially seen in the loss, which is 7.5470 for the first round, while in the previous cases this value was 5.5918 and 6.6244 respectively. This is because the third client entered the training when there were already 2 clients training the first round, so this client did not train the model in that round but was used during the aggregated evaluation. The rest of the training continues with the 3 clients, with an anomaly in round 5, where there is a worsening in both metrics that is corrected in the next round. Specifically, the impact of removing a client from the training is clearly seen in round 10 (the loss increases from 0.0572 to 0.5715 and the accuracy decreases from 0.9853 to 0.9084). From the model performance point of view, this was as expected, and from the computational one, the elimination of this client from the training has not impacted the process, which has continued normally (showing that no up-dates have been received from one of the clients in round 9), since the minimum number of clients both available and to fit was set to 2.

5 CONCLUSIONS AND FUTURE DIRECTIONS

This paper presents the implementation of a complete federated learning architecture implemented on the AI4EOSC platform using flower. The platform allows users to request the computational resources needed for their applications and can use these resources, including the availability of GPUs, to train the models from the clients' side.

The platform's intuitive interface allows to deploy in a few steps the server with the desired IDE (it is recommended to use Jupyter Notebook or VS Code for better interaction and functionality). The FL server configuration is performed on the dashboard itself.

In addition, we introduce the implementation of the platform, the API and the dashboard with the focus on the FL server as a tool for the latter. In addition, modifications made to the Flower Python library in order to connect the clients when the server runs behind a proxy (in this case Traefik), and to allow client authentication when connecting to the server are explained. In the last case, a service for secret management has also been deployed.

Finally, the application of FL to a medical imaging use case is demonstrated with three clients simulating three hospitals. Specifically, the three clients run and store their data on three different OpenStack cloud instances. The implementation of the architecture applied to this use case is shown step-by-step through several code snippets. Finally, the impact on the configuration and development of the training with the server running on the AI4EOSC platform is analyzed in the case of intermittent clients; both if one joins the training once it has already started and if a client leaves it.

Regarding *future directions*, there is a lot of work that can be done in the near future to support the use of privacy-aware distributed AI, especially concerning federated learning applications in the EOSC. An example of such support of the research infrastructure is training scalability or a research approach to reduce computational and communication overhead. From a data privacy point of view, the possibility of supporting other PETs is also still open, in addition to provide users with the possibility of validating the efficiency of their privacy-enhancing solutions using techniques for attacking FL. Furthermore, explore ways of preventing poisoning attacks when authenticated clients misbehave and send anomalous updates. Finally, it is key to allocate even more high memory resources (RAM or GPUs) for simulations of more clients, higher computation requirements for encryption operation testing, and cooperation with the security infrastructure to ensure integrity and confidential computing in the mean of decentralized data security.

ACKNOWLEDGMENTS

The authors would like to thank the funding through the project AI4EOSC "Artificial Intelligence for the European Open Science Cloud" that has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement number 101058593.

CRediT authorship contribution statement

Judith Sáinz-Pardo Díaz: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Software, Visualization, Writing - Original Draft. **Andrés Heredia Canales**: Conceptualization, Methodology, Software, Writing - Original Draft. **Ignacio Heredia Cachá**: Conceptualization, Methodology, Software, Writing - Original Draft. **Viet Tran**: Formal Analysis, Validation, Software, Writing - Review and Editing. **Giang Nguyen**: Formal Analysis, Methodology, Validation, Writing - Review and Editing. **Khadijeh Alibabaei**: Investigation, Writing - Original Draft. **Marta Obregón Ruiz**: Software, Writing - Original Draft. **Susana Rebolledo Ruiz**: Software, Writing - Original Draft. **Álvaro López**

García: Conceptualization, Funding Acquisition, Project Administration, Methodology, Resources, Software, Supervision, Writing - Original Draft.

REFERENCES

[1] [n. d.]. API Documentation & Design Tools for Teams | Swagger — swagger.io. https://swagger.io/. [Accessed 01-02-2024].

[2] [n. d.]. Vault by HashiCorp — vaultproject.io. https://www.vaultproject.io/. [Accessed 29-02-2024].

[3] Sawsan AbdulRahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. 2021. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE INTERNET OF THINGS JOURNAL* 8, 7 (APR 1 2021), 5476–5497. https://doi.org/10.1109/JIOT.2020.3030072

[4] AI4EOSC. 2024. GitHub - AI4EOSC/flower: Adaptations of the flower framework for use in AI4EOSC. — github.com. https://github.com/AI4EOSC/flower/tree/credentials [Accessed 29-02-2024].

[5] AI4EOSC. 2024. GitHub - ai4os/ai4-flwr: AI4OS extensions for the Flower framework. — github.com. https://github.com/ai4os/ai4-flwr/tree/develop [Accessed 29-02-2024].

[6] AI4EOSC. 2024. GitHub - ai4os/ai4-papi: A Python library for interacting with the AI4EOSC services. — github.com. https://github.com/ai4os/ai4-papi [Accessed 12-01-2024].

[7] AI4EOSC Consortium. 2024. *AI4EOSC Project.* https://ai4eosc.eu/ [Accessed 15-01-2024].

[8] Syreen Banabilah, Moayad Aloqaily, Eitaa Alsayed, Nida Malik, and Yaser Jararweh. 2022. Federated learning review: Fundamentals, enabling technologies, and future applications. *Information processing & management* 59, 6 (2022), 103061.

[9] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. 2020. Flower: A Friendly Federated Learning Research Framework. *arXiv preprint arXiv:2007.14390* (2020).

[10] R Cook, W Michener, D Vieglais, A Budden, and R Koskela. 2012. Dataone: A distributed environmental and earth science data network supporting the full data life cycle. In *EGU General Assembly Conference Abstracts*. 11863.

[11] CYVERSE. 2024. CyVerse: Cyberinfrastructure for Life Sciences. https://cyverse.org/ [Accessed 16-01-2024].

[12] Kang Zhang Daniel Kermany and Michael Goldbaum. 2018. Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification. Mendeley Data, V2. https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia/code. https://doi.org/10.17632/rscbjbr9sj.2

[13] EGI. 2024. EGI: Advanced Computing for a Data-Driven Future. https://www.egi.eu/ [Accessed 15-01-2024].

[14] Mohammed El Hanjri, Hibatallah Kabbaj, Abdellatif Kobbane, and Amine Abouaomar. 2023. Federated Learning for Water Consumption Forecasting in Smart Cities. In *ICC 2023 - IEEE International Conference on Communications*. 1798–1803. https://doi.org/10.1109/ICC45041.2023.10279576

[15] EOSC. Year Accessed. European Open Science Cloud (EOSC) Portal. https://eosc-portal.eu/ [Accessed 16-02-2024].

[16] FATE. 2024. FATE (Federated AI Technology Enabler). https://github.com/FederatedAI/FATE [Accessed 09-01-2024].

[17] Yujia Gao, Liang Liu, Binxuan Hu, Tianzi Lei, and Huadong Ma. 2020. Federated Region-Learning for Environment Sensing in Edge Computing System. *IEEE Transactions on Network Science and Engineering* 7, 4 (2020), 2192–2204. https://doi.org/10.1109/TNSE.2020.3016035

[18] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. 2003. Terra: A Virtual Machine-Based Platform for Trusted Computing. *SIGOPS Oper. Syst. Rev.* 37, 5 (oct 2003), 193–206. https://doi.org/10.1145/1165389.945464

[19] Google. 2024. Angular. https://angular.io/ [Accessed 26-01-2024].

[20] Google. 2024. Angular Material UI Library. https://material.angular.io/ [Accessed 26-01-2024].

[21] Google. 2024. Google Colaboratory. https://colab.google/ [Accessed 15-01-2024].

[22] Google. 2024. Release-Please project. https://github.com/googleapis/release-please [Accessed 26-01-2024].

[23] Harbor. 2024. Harbor registry. https://goharbor.io/ [Accessed 26-01-2024].

[24] HashiCorp. 2024. Consul | HashiCorp Developer — developer.hashicorp.com. https://developer.hashicorp.com/consul [Accessed 15-01-2024].

[25] HashiCorp. 2024. Nomad | HashiCorp Developer — developer.hashicorp.com. https://developer.hashicorp.com/nomad [Accessed 12-01-2024].

[26] Jest. 2024. Jestjs testing library. https://jestjs.io/ [Accessed 26-01-2024].

[27] jrxFive. [n. d.]. GitHub - jrxFive/python-nomad: Client library Hashicorp Nomad — github.com. https://github.com/jrxFive/python-nomad. [Accessed 01-02-2024].

[28] Jupyter. 2024. JupyterLab: A Next-Generation Notebook Interface. https://jupyter.org/ [Accessed 15-01-2024].

[29] Jakub Konecný, H. B. McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *ArXiv* abs/1610.05492 (2016). https://api.semanticscholar.org/CorpusID:14999259

[30] Traefik Labs. 2024. Traefik Proxy | Traefik Labs — traefik.io. https://traefik.io/traefik [Accessed 15-01-2024].

[31] Junqing Le, Di Zhang, Xinyu Lei, Long Jiao, Kai Zeng, and Xiaofeng Liao. 2023. Privacy-Preserving Federated Learning With Malicious Clients and Honest-but-Curious Servers. *IEEE Transactions on Information Forensics and Security* 18 (2023), 4329–4344. https://doi.org/10.1109/TIFS.2023.3295949

[32] Álvaro López García, Jesús Marco De Lucas, Marica Antonacci, Wolfgang Zu Castell, Mario David, Marcus Hardt, Lara Lloret Iglesias, Germán Moltó, Marcin Plociennik, Viet Tran, Andy S. Alic, Miguel Caballer, Isabel Campos Plasencia, Alessandro Costantini, Stefan Dlugolinsky, Doina Cristina Duma, Giacinto Donvito, Jorge Gomes, Ignacio Heredia Cacha, Keiichi Ito, Valentin Y. Kozlov, Giang Nguyen, Pablo Orviz Fernández, Zděnek Šustr, and Pawel Wolniewicz. 2020. A Cloud-Based Framework for Machine Learning Workloads and Applications. *IEEE Access* 8 (2020), 18681–18692. https://doi.org/10.1109/ACCESS.2020.2964386

[33] Microsoft. 2024. Visual Studio Code. https://code.visualstudio.com/ [Accessed 15-01-2024].

[34] Dianwen Ng, Xiang Lan, Melissa Min-Szu Yao, Wing P Chan, and Mengling Feng. 2021. Federated learning: a collaborative effort to achieve better medical imaging models for individual sites that have small labelled datasets. *Quantitative Imaging in Medicine and Surgery* 11, 2 (2021), 852.

[35] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, Dusit Niyato, and H Vincent Poor. 2021. Federated learning for industrial internet of things in future industries. *IEEE Wireless Communications* 28, 6 (2021), 192–199.

[36] Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. 2020. Federated learning for wireless communications: Motivation, opportunities, and challenges. *IEEE Communications Magazine* 58, 6 (2020), 46–51.

[37] The Open Science Grid Executive Board on behalf of the OSG Consortium: Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, Ian Foster, Rob Gardner, Mike Wilde, Alan Blatecky, John McGee, and Rob Quick. 2007. The Open Science Grid. *Journal of Physics: Conference Series* 78, 1 (July 2007), 012057. https://doi.org/10.1088/1742-6596/78/1/012057

[38] OpenMined. 2024. OpenMined/PySyft. https://github.com/OpenMined/PySyft [Accessed 09-01-2024].

[39] AI4EOSC project. 2023. AI4EOSC Flower extensions. https://github.com/AI4EOSC/flower/tree/develop [Accessed 09-01-2024].

[40] AI4EOSC project. 2024. AI4EOSC Architecture repository. https://github.com/AI4EOSC/ai4-architecture [Accessed 15-01-2024].

[41] AI4EOSC project. 2024. AI4EOSC dashboard (GitHub). https://github.com/ai4os/ai4-dashboard [Accessed 15-01-2024].

[42] AI4EOSC project. 2024. AI4EOSC dashboard (platform). https://dashboard.cloud.ai4eosc.eu/ [Accessed 15-01-2024].

[43] AI4EOSC project. 2024. Federated Learning Server. https://github.com/deephdc/federated-server [Accessed 09-01-2024].

[44] Sebastián Ramírez. [n. d.]. *FastAPI.* https://github.com/tiangolo/fastapi

[45] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. 2020. The future of digital health with federated learning. *NPJ digital medicine* 3, 1 (2020), 119.

[46] Holger R Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, et al. 2022. Nvidia flare: Federated learning from simulation to real-world. *arXiv preprint arXiv:2210.13291* (2022).

[47] Judith Sáinz-Pardo Díaz, María Castrillo, and Álvaro López García. 2023. Deep learning based soft-sensor for continuous chlorophyll estimation on decentralized data. *Water Research* 246 (2023), 120726.

[48] Micah J Sheller, Brandon Edwards, G Anthony Reina, Jason Martin, Sarthak Pati, Aikaterini Kotrotsou, Mikhail Milchenko, Weilin Xu, Daniel Marcus, Rivka R Colen, et al. 2020. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific reports* 10, 1 (2020), 12598.

[49] Judith Sáinz-Pardo Díaz and Álvaro López García. 2023. Study of the performance and scalability of federated learning for medical imaging with intermittent clients. *Neurocomputing* 518 (2023), 142–154. https://doi.org/10.1016/j.neucom.2022.11.011

[50] TF-Federated. 2024. Tensorflow Federated: Machine Learning on Decentralized Data. https://www.tensorflow.org/federated [Accessed 09-01-2024].

[51] Guido Van Rossum and Fred L Drake Jr. 1995. *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam.

[52] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. 2020. A survey on distributed machine learning. *Acm computing surveys (csur)* 53, 2 (2020), 1–33.

[53] Runhua Xu, Nathalie Baracaldo, and James Joshi. 2021. Privacy-Preserving Machine Learning: Methods, Challenges and Directions. arXiv:2108.04417 [cs.LG]